ACNAUX Functions

Acnet utility support

Mar 6, 1992

Introduction

The Acnet standard for task–task communications is widely used in accel erator control systems at Fermilab. The Acnet task called ACNAUX is designed to sup port a set of utility functions which can aid diagnosis of Acnet network nodes in a standard way, as it is independent of the cpu and operating system used by each node. Each node supports the Acnet header standard; through ACNAUX each node supports some common diagnostic utility functions. This note des cribes the support for ACNAUX in the Local Station nodes. The official document of the standard is described elsewhere by Glenn Johnson.

Each function is specified by the lo byte of the first (or only) word following the Acnet header of the request message. In some cases the hi byte of this word may be used as a sub-function code. All functions are one-shot requests.

ACNAUX implementation

In the local station, ACNAUX is implemented by a local application called AAUX. It uses the generic protocol support available via OpenPro to receive notification about network messages directed to it. This same support is also used by FTPMAN, GATE, and HUMBUG. It permits more rapid response than would be achieved using 15 Hz polling of the message queue. Two of the available parameter words are used to pass a ptr to the message reference block that itself includes a ptr to the received request message. The AcReq Task calls the local application when it receives a message destined for the local application whose network task name is found in the protocol table, filled by OpenPro calls.

NOOP function 0

This serves as a “ping” facility. It determines if a node will respond to an Acnet header request message. A status-only reply is returned. A Vax program called ANPING can be run from a terminal to exercise this function. It includes the time for the response in 10 msec resolution. A local station which is not busy can return such a response in 4 msec, which is near the limit of the token ring chipset that interfaces to the token ring network.

GTTASK function 4

Returns a list of the currently-connected network task names, followed by a byte array of the associated task-id’s. The AAUX local application examines the NETCT table contents to find this info. For each entry whose queue id is nonzero, the task name and id is recorded. Because the format is specified in Vax normal byte order, it is necessary to swap bytes for all words in the reply.

The byte order of task names used in the local stations was designed to conform to the notion that a task name can be a 4-byte character array. But in the acnet system, many task names are in 6-character RAD–50 format, which also takes 4 bytes (two 3-character words). (Recall for the following argument that the token ring hardware interface on the Vax swaps every byte, in order to make it such that 2-byte integer words transfer between Vax and token ring stations that use a "big-endian" architecture without software byte swapping.) To make it possible for both the Vax and the local station to use 4-character ascii names, the bytes of the destination task name field of an acnet header are swapped upon reception by the ANet Task in the local station. ANet then searches for a match with the current connected task entries in the NETCT net connection table to dispatch the received message to the proper message queue. When a message is transmitted by the local station, these bytes are swapped before it gets passed to the chipset so that the Vax receives them in natural order.

As a result of this logic of preserving 4-char ascii task name communication, the 6-character RAD–50 names must be kept in byte-swapped form in the NETCT table. Since these names are treated as magic constants by local station software, this is easy to do. As an example, the task name ACNAUX in RAD–50 form is $06C609A0 (ACN=06C6, AUX=09A0); but for local station software, it should be specified as $C606A009, and it appears this way in the NETCT table entry.

Since there can be a mix of 4-character and 6-character formats, it requires some special logic to convert the names to ascii for display. All 4-character task names are composed of 4 capital letters in ascii. If a given task name fits this pattern, then it may be presumed a 4-character form; otherwise, it should be assumed to be of the 6-character RAD–50 form.

RAD–50 definition

This encoding of a restricted set of characters permits squeezing 3 characters of information into one 16-bit word. It can be considered simply as a base-40 number system, whose coding scheme is as follows:

0	space	28	.
1–26	A–Z	29	(unused)
27	$	30–39	0–9

To convert 'XYZ' into RAD–50, the result is (25*40 + 26)*40 + 27 = 41067 = $A06B.

GTTRIO function 8

Returns token ring chipset I/O error statistics. The token ring chipset main tains an error log that is a set of nine 8-bit counters. A special command can be issued to the chipset to interrogate these counters. An extra motivation for doing so is provided by the fact that for some error conditions, when the error count reaches 255, or $FF, the chipset removes itself from the network. This means that a node on token ring should plan to read this error log on some periodic basis. The local station software does this, using a default period of about 20 minutes,

currently. The counts are accumulated into a corresponding set of 16-bit counts, which allow monitoring the health of the network. This GTTRIO function returns the value of these word counts, along with the time interval over which they were accumulated. The names of the error conditions are:

Line

Each frame that is received or repeated for a valid FCS or Manchester code violation. If one is detected, the EDI (Error Detected Indicator) bit is set to "1" in the frame or token's ending delimiter. If the received EDI is "1", this Line error count is incremented; if the EDI is a "1", it is not incremented.

ARI/FCI

This indicates that the up-stream node chipset is unable to set its ARI/FCI bits in a frame it has received. (The details of this seem rather obscure to this writer.)

Burst

The chipset has detected the absence of transitions for five half-bit times between SDEL and EDEL.

Receive congestion

The chipset recognizes a frame addressed to its specific address, but it has no buffer space available to receive the frame.

Lost frame

When in transmit mode, the chipset fails to receive the end of the frame it has transmitted.

Frame copied

When in receive/repeat mode, the chipset recognizes a frame that is addressed to its specific address, but the ARI bits are nonzero, indicating a possible duplicate address. (The bridge currently causes many of these.)

Token

The Active Monitor detects a frame with the MONITOR COUNT bit set, no token of frame received within a 10 msec window, or a code violation in a starting delimiter/token sequence.

DMA parity or *DMA Bus*

Maybe something wrong with the token ring interface board itself.

GTPKTS function 9

Returns network message packet processing statistics to permit assessment of a node's network I/O activity. The time since the network statistics were cleared is given along with a count of message packets processed either in or out. For the local station, several resident diagnostic counters are monitored to collect these statistics. The time period is the time since the AAUX local application was last initialized, which would normally be at system reset time; however, if AAUX is updated to a new version, upon download of a new version, the old version is terminated and the new version is initialized, so the statistics will begin again. The implementation uses the cycle counter which is a longword that begins at

zero at system reset time and is incremented for every 15 Hz cycle. If a station is running at the backup 12.5 Hz rate, this value is not corrected for it. When AAUX is initialized, it captures the present reading of this cycle counter. To reply to a GTPKTS function request, the current cycle counter – the earlier one is returned.

The count of packets processed is fairly involved. The local station supports network communications with several protocols. The Classic data request / alarm pro to col does not use an acnet header. The DZero and Accelerator protocols do use an acnet header. Each family of protocols must be considered separately.

The Classic protocol uses SAP 18. At present, there is no message counter accumulated for the Classic protocol messages received, so a frame counter is monitored as an approximation. In the SAP table, a word counter is incremented for each SAP 18 frame received. AAUX watches this counter every cycle and notices changes in it to build a count of Classic frames received. When a message count is added to the system logic, this code can be updated to use it.

All Acnet-header protocols use SAP 68. Each task name that is connected to the network is recorded in the NETCT table, and a word counter is incremented for each message that is received and dispatched by the ANet Task. So AAUX monitors these counters for all 23 possible entries in NETCT. For each entry that is active (queue id <> 0) the associated word counter is monitored every cycle for evidence of counting. Increments are accumulated into the total packet count.

All messages transmitted pass through the OUTPQ network output pointer queue. There is a word in the OUTPQ header that is incremented for every message that has been completely transmitted. This word is monitored every cycle and any increments noticed are accumulated into the total packet count.

All in all, the total packet count is the sum of the number of Classic frames received (to be replaced by a message count when available), the number of messages passed to the associated message queue for each connected network task name, and the number of messages completely transmitted to the network for any protocol.

Each Linac local station uses Arcnet communications for data acquisition with the SRMs (Smart Rack Monitors), which usually number 4 or 5 on each Arcnet. This communication protocol is Acnet-header based and is called locally “#4” to signify it was the 4th data request protocol to be supported by the local station system software. This network activity is not part of the token ring network activity; therefore, even though it represents network processing activity in the local station, it was *not* included in the #packets reported in reply to GTPKTS. If it were, it would typically add 75 packets per second for a station with 4 SRMs. This includes 1 broadcast transmitted request and 4 replies per 15 Hz cycle.

Viewing the results of these functions

One program that makes use of the GTPKTS function are Vax console page D31, which polls a large sequence of nodes and reports the total time value and the number of network packets processed per second between polls. Any node that does not support the GTPKTS function is sent a NOOP function instead, in which case only an indication of the success of the reply is reported.

Another program that exercises this function is called PACKETS, accessible from a VT100 terminal or emulator. It shows the network activity relating to 4 nodes, with the last one initially set to ADCALC. This last one can be changed to a user selected node name by typing "!" to get the prompt message that asks for the node name, such as LIN611, for example. When a user-selected name is entered, the GTTASK function is issued to request the task list, displayed in a separate box. Each node is polled approximately every 4 seconds, and statistics are displayed for the current packet rate, the minimum and maximum rates, and rates that are averaged over several different time intervals.

The program that uses the GTTRIO function to list the error counters at a VT100 terminal is called TRIO. It prompts for a node name and shows the current counts obtained via the GTTRIO function and also the time over which the counts were accumulated. At this time, the CLEAR and NOW and set new period are not implemented, but they could be so in the future if needed.